R

FG

COMPUTER SCIENCE TECHNICAL REPORT SERIES

UNIVERSITY OF MARYLAND
COLLEGE PARK, MARYLAND
20742

(12)

(9) Technical Re[redacted] TR-394 (14)
~~NSF-OCA-GJ-35568X-394~~
(15) [redacted]N00014-67-A-0239-0021
NSF GJ-35568

(11) Aug[redacted]975

(16) NR-044-431

(6)

Further Programs for the Solution of Large Sparse Systems of Linear Equations.

by

Charles Werner

(10) [redacted] K. Mesztenyi [redacted] C. Rheinboldt

(12) 51 p.

Abstract

A package of FORTRAN subroutines is presented for the solution of nonsymmetric or symmetric sparse linear systems by triangular decomposition. Two principal aims are (1) to handle matrices which originally fit into primary core storage but do so no longer after decomposition, and (2) to solve a sequence of linear systems all of which have the same sparsity structure by generating--in secondary storage--a record of the decomposition process in the form of an integer array. Some experimental results using the package are included.

FURTHER PROGRAMS FOR THE SOLUTION OF
LARGE SPARSE SYSTEMS OF LINEAR EQUATIONS[1)]

by

Charles K. Mesztenyi[2)] and Werner C. Rheinboldt[2)]

1. Introduction

In a previous report [1] a package of FORTRAN subroutines was presented for the solution of a linear system

$$Ax = b \tag{1}$$

based on triangular decomposition of the (symmetric or nonsymmetric) matrix A. The underlying data structure was motivated by a more general arc-graph structure discussed in [2].

The programs given here have the same purpose but pursue the following two different aims:

a. To handle matrices which originally fit into primary core storage but do so no longer after decomposition.

b. To solve a sequence of systems (1) all of which have the same sparsity structure. This case arises, for example, in the solution of nonlinear systems by Newton's method.

The first aim is accomplished during decomposition by writing the decomposed part of the matrix into secondary storage and using its place for newly introduced nonzero elements. In order to meet the second aim

---

1) This work was supported in part by the National Science Foundation under Grant GJ-35568X and the Office of Naval Research under Grant N00014-67-A -0239-0021 (NR-044-431).

2) Computer Science Center, University of Maryland, College Park, Md. 20742.

we follow an idea in [3] and generate--in secondary storage--a record of the decomposition process in the form of an integer array. This record can be used to decompose any matrix with the same sparsity structure provided there are no round-off problems.

2. Some Background

The desired triangular decomposition of the $n \times n$, nonsingular matrix A has the form

$$(2) \qquad PAQ = LU, \quad L = I + L^0 ,$$

where L^0 is strictly lower triangular, U upper triangular, and the permutation matrices P,Q define the pivoting sequence. The decomposition is accomplished in n steps, such that

$$(3) \qquad P_iAQ_i = (I+L_i^0)U_i + A_i, \quad i = 0,1,\ldots,n$$

where the first i rows and i columns of A_i, the last $n-i$ columns of L_i^0, and the last $n-i$ rows of U_i are zero, respectively. Moreover, $PP_i^TL_i^0$ has the same first i columns as L^0 and $U_iQ_i^TQ$ the same first i rows as U. This latter fact allows us to keep L_i^0 and U_i in secondary storage.

Let $\eta(B)$ denote the number of nonzero elements of any matrix B. Then the storage required before and after decomposition is of the order of $m_0 = \eta(A)$ and $m_2 = \eta(U) + \eta(L^0)$, respectively. Furthermore, $m_1 = \max_i \eta(A_i)$ is the maximal storage needed for the matrices A_i. Clearly, we have

$m_0 \leq m_1 \leq m_2$ and, in practice, it turns out that $m_2 - m_0$ is very much larger than $m_1 - m_0$. In fact, sometimes we found m_1 to be equal to m_0 (see Section 5). Hence by retaining only the A_i in primary storage we require, in general, only little more storage than for A itself.

The basic storage structure allows for easy modification of the pivoting strategy. In fact, in the nonsymmetric case the pivot selection is handled by an easily replacable subroutine. We use here the well-known minimal degree algorithm. If S_i is the set of nonzero elements of A_i, then for any $x \in S_i$ we denote by $R_i(x)$ and $C_i(x)$ the subsets of S_i consisting of the elements in the same row and column as x, respectively. Now, with $E_i(x) = R_i(x)$ if $|R_i(x)| \leq |C_i(x)|$ and otherwise $E_i(x) = C_i(x)$, the set of potential pivots is given by

$$S_i^0 = \{x \in S_i;\ |a(x)| \geq \mu \max_{z \in E_i(x)} |a(z)|\}. \tag{4}$$

Here $a(z)$ is the value of the matrix element corresponding to z and $\mu \in [0,1]$, a user-defined parameter. The ith pivot is then the element $x \in S_i^0$ for which $(|R_i(x)|-1)(|C_i(x)|-1)$ is minimal. Generally, with decreasing μ the fill-in decreases while the round-off influence increases.

For symmetric A it is assumed that the pivots remain on the main diagonal and hence that $Q = P^T$. In that case each matrix A_i is again symmetric. If D_i is the set of nonzero diagonal elements of A_i, then the ith pivot is the element x of the set

$$D_i^0 = \{z \in D_i;\ |a(z)| \geq \mu \max_{y \in D_i} |a(y)|\} \tag{5}$$

for which the number of nonzero elements in its row is minimal.

It is theoretically possible to use the value $\mu = 0$. In that case, (4) and (5) show that any nonzero element of S_i or D_i, respectively, is a potential pivot. Then the pivot selection depends only on the sparsity structure and not on the elements of the matrix--but, of course, the round-off influence may be considerable.

3. Basic Storage Arrangements

3.1 Matrices in Primary Storage: As mentioned before, the basic storage structure used here is the same as that in [1]. We summarize it briefly.

Set $N = \{1,2,\ldots,n\}$ and let $S \subset N \times N$ be the set of locations corresponding to the nonzero elements of a given $n \times n$ matrix A. We number the elements of S from $n+1$ to $n+m$, $m = |S|$, that is, we introduce a bijective mapping

$$\nu: S \to \{n+1,\ldots,n+m\} \ . \tag{6}$$

Now define two integer arrays RY and CY each of length $n+m$ in which the relative addresses $n+1,\ldots,n+m$ correspond to the elements of S in the order provided by ν. The images $\nu(R_i)$ of the row sets

$$R_i = \{(i,k) \in S;\ \text{some } k \in N\},\ i \in N$$

form a partition of $\{n+1,\ldots,n+m\}$. For any set $\nu(R_i) = \{i_1,\ldots,i_k\}$ we link the locations $i, i_1, i_2, \ldots, i_k$ into a circular list

$$i_1 = RY(i),\ i_{j+1} = RY(i_j),\ j = 1,\ldots,k-1,\ i = R(i_k) \tag{7}$$

where for practical reasons

(8) $$i_1 > i_2 > \dots > i_k > i \, .$$

Analogously, we proceed with the images $\nu(C_i)$ of the column sets

$$C_j = \{(k,j) \in S; \text{ some } k \in N\}, \; j \in N$$

in the array CY.

In order to store the associated matrix elements a third array A is, of course, needed. Thus, for example, the matrix

$$\begin{pmatrix} 0 & 1 & 0 & 3 \\ 0 & 0 & 5 & 0 \\ -1 & 0 & 2 & 0 \\ 0 & -2 & 0 & 0 \end{pmatrix}$$

may be stored as follows:

loc	1	2	3	4	5	6	7	8	9	10
RY	10	8	9	7	3	1	4	2	5	6
CY	5	7	9	10	1	2	6	3	8	4
A	*	*	*	*	-1	1	-2	5	2	3

We shall refer to RY and CY as the sparsity structure arrays and to A as the coefficient array.

For symmetric A the set S only needs to be the set of locations of the nonzero elements in the upper (or lower) triangle (including the diagonal) of A. Moreover, we assume always that in the symmetric case all diagonal elements are nonzero. Then the first n cells of the sparsity structure arrays RY and CY are no longer needed if (6) is changed to

$$\nu\colon S \to \{1,2,\ldots,m\},\ \nu(i,i) = i, i=1,\ldots,n\ .$$

There is no need to repeat the details; the resulting data arrangement should be self-evident from the following example:

$$\begin{pmatrix} 1 & 0 & -1 & 0 \\ 0 & 2 & 0 & -2 \\ -1 & 0 & 3 & 0 \\ 0 & -2 & 0 & 4 \end{pmatrix}$$

loc	1	2	3	4	5	6
RY	5	6	3	4	1	2
CY	1	2	5	6	3	4
A	1	2	3	4	-1	-2

During decomposition, this storage arrangement is used for the matrices A_i, $i = 0,\ldots,n$. When a nonzero element of A_i remains in A_{i+1} its position in the RY,CY,AY arrays is maintained. After each pivoting step the pivot row and column are written on secondary storage and the corresponding cells in the storage arrays are freed, that is, the circular linkages containing these elements are modified appropriately. The resulting free locations are reassigned when fill-in occurs.

3.2 Triangular Matrices in Secondary Storage: The programs here are written for use with a random-access secondary storage device. Some information about the necessary I/O routines is provided in Section 4.1 below.

In the nonsymmetric case the triangular matrices L and U are written as two arrays of pairs of numbers. The L-array contains the columns of L in consecutive order and each column has the form

$$\begin{array}{c} -i_c, -i_r \\ j_1, \ell_{j_1, i_c} \\ \vdots \\ j_k, \ell_{j_k, i_c} \end{array}$$

where i_c and i_r are the column- and row-index, respectively, of the pivot (stored negatively) and j_i represents the row index and ℓ_{j_i,i_c} the value of each nonzero element in the particular column. Similarly the U-array contains the rows of U in consecutive order, each of them in the form

$$
\begin{array}{c}
j_1, u_{i_r,j_1} \\
\vdots \\
j_k, u_{i_r,j_k} \\
-i_c, p_{i_c}
\end{array}
$$

Here i_c is the column index of the pivot and p_i its value, while j_i, u_{i_r,j_i} denote the column index and value, respectively, of each nonzero element in the row. The entire U-array is initialized by a dummy pair -1,-1.

For the backsubstitution programs it is assumed that the L-array is read forward and the U-array backward.

In the symmetric case, there is, of course, no need for both the L- and U-array. Accordingly, only the L-array is set up containing the columns of L in consecutive order, each one in the form

$$
\begin{array}{c}
-i_d, d_{i_d} \\
j_1, \ell_{j_1,i_d} \\
\vdots \\
j_k, \ell_{j_k,i_d} \\
-i_d, d_{i_d}
\end{array}
$$

Here i_d is the index of the pivot (on the diagonal) and d_i its value and j_i, ℓ_{j_i}, i_d have the previous meaning. The header at the beginning and end of each column is needed, since during backsubstitution the array is read once forward and once backward.

3.3 Representation of the Decomposition Record: As mentioned in the introduction, the programs can generate a record of the decomposition process for later use with any other matrix of the same sparsity type. This record is in the form of an array of positive integers in secondary storage. For each pivoting step the following information is recorded:

Nonsymmetric Case:

$$\begin{aligned} &i_x,\ i_c,\ i_r,\ k_c,\ x_1,\ j_1,\ \cdots,\ x_{k_c},\ j_{k_c} \\ &\qquad k_r,\ y_1,\ m_1,\ \cdots,\ y_{k_r},\ m_{k_r} \\ &\qquad \ell_1,\ \ell_2,\ \cdots,\ \ell_t \end{aligned}$$

i_x relative location of the pivot in the RY,CY arrays

i_c, i_r the column- and row-index of the pivot, respectively

k_c, k_r the number of nonzero elements in the pivot-column and pivot-row, respectively

x_i, j_i relative location (in CY) and row-index, respectively, of the nonzero elements in the pivot column

y_i, m_i relative location (in RY) and column-index, respectively, of the nonzero elements in the pivot row

ℓ_i relative locations of the elements in A_i which must be modified at the step, $t = k_c \cdot k_r$

Symmetric Case:

$$i,\ k,\ y_1,\ m_1,\ \ldots\ ,\ y_k,\ m_k$$
$$\ell_1,\ \ldots\ ,\ \ell_t$$

i index of the pivot and hence also its relative location in the RY,CY arrays

k the number of nonzero elements in the pivot row. Each of these elements is identified by a pair (y_j, m_j) as in the nonsymmetric case

ℓ_j the relative locations of the elements in A_i to be modified, $t = k(k-1)$

4. Description of the Programs

The package consists of four groups of subroutines with names INT, BLD, DEC, and SLV; in addition, there is a pivot selection routine PVT01 for the nonsymmetric case and a set of I/O routines for interface with the random storage device.

The INT programs initialize the storage area and have to be called first. The BLD routines establish the data structure described in Section 3.1 for the given matrix A. Then the DEC routines are called to perform the decomposition of the matrix and/or to generate a record of the decomposition process. Finally, if applicable, the SLV routines are used to obtain the solution of the given system (1) by backsubstitution.

In general, any efficient routine for building up the basic data structure from given data about the matrix depends strongly on the details

of the files used. The BLD programs presented here avoid all assumptions about file formats, etc., by establishing the data structure one matrix element at a time. In other words, the chosen BLD routine has to be called once for each nonzero matrix element. For many practical purposes this may be inefficient. The routines were included principally for the sake of completeness; it should be easy to rewrite them for any specific application.

The names of all subroutines in the four principal groups are preceded by the letters S or N for the case of symmetric or nonsymmetric matrices, respectively. The names of the subroutines in the INT, BLD, DEC group are ending with one of the numerals 0, 1 or 01. This indicates the following alternatives:

0 - Initialize or build only the sparsity structure arrays of the matrix or generate a record of the decomposition based solely on the sparsity structure. These routines are only available for symmetric matrices; for nonsymmetric matrices it is not an advisable approach since the resulting round-off error could be severe.

1 - Initialize or build only the coefficient array for the matrix elements, or decompose the matrix using a previously generated record of a decomposition for matrices with the same sparsity structure.

01 - Initialize or build both the sparsity structure arrays and the coefficient array, or decompose the given matrix and, optionally, generate a record of the decomposition.

The pivot selection for the nonsymmetric case is performed by the routine PVT01. For the symmetric case, pivot selection is incorporated within the routines SDEC0 and SDEC01.

4.1 Catalog of Subroutines: In this subsection we list the various subroutines of the package together with their calling sequences and brief descriptions of their purposes. The arguments in the calling sequences are discussed in the next subsection.

INT - Routines

SINT0(MD,RY,CY,ND)

Initialize the sparsity structure arrays of a symmetric matrix.

SINT01(MD,FD,RY,CY,A,AN,ND)

Initialize the sparsity structure arrays and the coefficient array of a symmetric matrix.

SINT1(MD,FD,AN)

Initialize the coefficient array of a symmetric matrix.

NINT01(MD,FD,RY,CY,AN,NDR,NDC)

Initialize the sparsity structure arrays and the coefficient array of a nonsymmetric matrix.

NINT1(MD,FD,AN)

Initialize the coefficient array of a nonsymmetric matrix.

BLD - Routines

All routines add a matrix element with value V, row index I, and column index J to the structure. Note that in the symmetric case only the nonzero elements in the upper (or lower) triangle and the diagonal should be given.

SBLD0(I,J,MD,RY,CY,ND)

Insert element (I,J) into the sparsity structure arrays of a symmetric matrix.

SBLD01(I,J,V,MD,FD,RY,CY,A,AN,ND)

Insert element (I,J) into the sparsity structure arrays of a symmetric matrix and a corresponding value V into the coefficient array.

SBLD1(I,J,V,MD,FD,A,AN)

Associate a value V to element (I,J) of a symmetric matrix. The V-values must be in the same order in which the (I,J)-values were read-in during prior construction of the corresponding sparsity structure arrays by SBLD0 or SBLD01.

NBLD01(I,J,V,MD,FD,RY,CY,A,AN,NDR,NDC)

Insert element (I,J) into the sparsity structure arrays of a nonsymmetric matrix and a corresponding value V into the coefficient array.

NBLD1(I,J,V,MD,FD,A,AN)

Associate a value V to element (I,J) of a nonsymmetric matrix. The V-values must be in the same order in which the (I,J)-values were read-in during prior construction of the corresponding sparsity structure arrays by NBLD01.

DEC - Routines

SDEC0(MD,RY,CY,ND,IP,IE,IH)

Generate a record of the decomposition of a symmetric matrix on the basis of the given sparsity structure.

SDEC01(MD,FD,RY,CY,A,AN,ND,IP,IE,IH)

Decompose a given symmetric matrix and optionally (MD(3)≠0) generate a record of the decomposition.

SDEC1(MD,FD,A,AN,IE)

Decompose a given symmetric matrix using a previously generated decomposition record.

NDEC01(MD,FD,RY,CY,A,AN,NDR,NDC,IPR,IPC,IE,IH,NG1,NG2)

Decompose a given nonsymmetric matrix and optionally (MD(3)≠0) generate a decomposition record.

NDEC1(MD,FD,A,AN,IE,IH)

Decompose a given nonsymmetric matrix using a previously generated decomposition record.

Pivot Routine

PVT01(I,N,IX,KR,KC,F,RY,CY,A,IPR,IPC,NDR,NDC,IE,IH)

Select the next pivot by the minimal degree algorithm during the decomposition of a nonsymmetric matrix. The routine is used by NDEC01.

SLV - Routines

These routines use the decomposed matrix in secondary storage. The right side of the system is given in the form of the input array X which in turn may be the same as the output array Y of the solution. The routines may be called repeatedly for different right sides.

SSLV(MD,X,Y)

Return the solution Y of the symmetric system with right side in X.

NSLV(MD,X,Y,AN)

Return the solution Y of the nonsymmetric system with right side in X.

I/O - Routines

The I/O routines for communication with the random access storage device are not in basic FORTRAN. They should be modified to suit a user's machine configuration. The routines have the following entries:

For I/O of decomposition record (array of positive integers)

DWI - Initialize for writing.

DW(K) - Write K as next entry of the array.

DWE - Terminate writing.

DRI - Initialize for reading.

DR(K) - Return the next entry of the array in K.

For I/O of symmetric decomposed matrix (array of pairs)

SVWI - Initialize for writing.

SVW(ℓ,s) - Write (ℓ,s) as next entry of the array. ℓ-signed integer, s-real.

SVWE - Terminate writing.

SVRI - Initialize for reading (forward).

SVRF(ℓ,s) - Return the next entry of the array in ℓ and s.

SVRB(ℓ,s) - Return the previous entry of the array in ℓ and s.

<u>For I/O of a nonsymmetric decomposed matrix</u> (two arrays of pairs)

NVWI - Initialize both files for writing.

NVWF(ℓ,s) - Write (ℓ,s) as next entry of the L-array (ℓ-signed integer, s-real).

NVWB(ℓ,s) - Write (ℓ,s) as next entry of the U-array.

NVWE - Terminate writing of both files.

NVRI - Initialize for reading, file L forward, file U backward.

NVRF(ℓ,s) - Return next entry of L-array in (ℓ,s).

NVRB(ℓ,s) - Return previous entry of U-array in (ℓ,s).

The following Table 1 shows the usage of the I/O routines by the various main routines:

Table 1

<u>I/O Routine Usage</u>

Program	DWR Decomposition Record File 10		SVWR Symmetric Dec. Matrix File 11		NVWR Nonsymmetric Dec. Matrix Files 12, 13	
	Write	Read	Write	Read	Write	Read
SDECO	X					
SDEC01	Ø		X			
SDEC1		X	X			
SSLV				X		
NDEC01	Ø				X	
NDEC1		X			X	
NSLV						X

X - routine used

Ø - use is optional, depending on user's request

4.2 <u>Arguments</u>: The arguments in the various calling sequences either reference single values or data arrays. For simplicity the single variables are collected in two arrays, an integer array

$$MD(I), \ I = 1,2,\ldots,8$$

and a real array

$$FD(I), \ I = 1,2,\ldots,7 \ .$$

The first three values of MD and the first two of FD are to be supplied by the user; the others represent output of various other routines. Care should be taken that these latter values are not modified whenever they are still to be used as input by other routines.

The use of the various arguments by the routines in the package is summarized in Tables 2 and 3 below.

<u>MD - Array</u>

MD(1) = N The dimension of the matrix; to be supplied by the user.

MD(2) = MX The lengths of the arrays RY, CY and A. If the decomposition of the matrix requires more internal storage, that is, if $m_1 > MX$, then the error indicator MD(4) is set to one and the process is terminated with a return to the user's main program.

MD(3) If this indicator is zero, the DEC01 program does not produce a decomposition record; for any nonzero value of MD(3) such a record is generated.

MD(4) Error indicator set as follows:

= 0 : no error.

= 1 : storage overflow; MX is too small for decomposition.

= 2 : on the basis of the sparsity structure (independent of the values of the elements) the matrix is singular.

= 3 : the matrix is declared numerically singular.

MD(5) In the nonsymmetric case equal to M0 + N where M0 is the number of nonzero elements in the matrix; in the symmetric case equal to M0, the number of nonzero elements on the diagonal and in the upper (or lower) triangle of the matrix.

MD(6) The length of the actually utilized portion of the arrays RY, CY or A, equal to M1 + N or M1 in the nonsymmetric or symmetric case, respectively.

MD(7) In the nonsymmetric case equal to M2 + N where M2 is the number of nonzero elements in the decomposed matrix; in the symmetric case equal to the nonzero elements on the diagonal and in the lower triangle after the decomposition.

MD(8) Length of the decomposition record.

FD - Array

FD(1) A tolerance EPS supplied by the user. If a pivot value in magnitude is less than EPS the matrix is considered to be numerically singular and the decomposition is terminated.

FD(2) Initial input by the user containing the pivot selection parameter μ.

FD(3) Largest coefficient value in magnitude in the original matrix. Initialized by the INT routines and updated by the BLD routines.

FD(4)	Largest coefficient value in magnitude encountered during decomposition, calculated by the DEC routines.
FD(5)	Natural logarithm of the absolute value of the determinant calculated by the DEC routines.
FD(6)	The sign of the determinant as +1.0 or -1.0 calculated by the DEC routines.
FD(7)	The natural logarithm of the product of the L_2 norms of the row vectors of the original matrix calculated by the DEC routines.

Data Arrays

RY(MX),CY(MX)	Integer arrays for the sparsity structure. Their length MX is specified by MD(2).
A(MX)	Real array for the values of the nonzero matrix elements.
AN(N)	Real array of length N (see MD(1)) used to collect row-vector norms of the matrix.
ND(N)	For symmetric A.
NDR(N),NDC(N)	For **nonsymmetric** A. Integer arrays of length N containing the number of nonzero elements by row (or column).
IE(N),IH(N)	For any matrix.
IP(N)	For symmetric matrices.
IPR(N),IPC(N) NG1(N),NG2(N)	For nonsymmetric matrices. Temporary arrays of length N.

Except for the last seven temporary arrays all data arrays are initialized in the appropriate INT routines. All integer arrays are used only for storing nonnegative integers. Thus for particular computers these arrays could be packed together.

Name	I	J	V	MD	FD	RY	CY	A	AN	ND	IP	IE	IH	X	Y
Type	I	I	R	I	R	I	I	R	R	I	I	I	I	R	R
Length / Program	1	1	1	8	7	M_1	M_1	M_1	N	N	N	N	N	N	N
SINT0	-	-	-	US	-	S	S	-	-	S	-	-	-	-	-
SINT01	-	-	-	US	US	S	S	S	S	S	-	-	-	-	-
SINT1	-	-	-	UPS	US	-	-	-	S	-	-	-	-	-	-
SBLD0	U	U	-	PS	-	PS	PS	-	-	PS	-	-	-	-	-
SBLD01	U	U	U	PS	PS	PS	PS	PS	PS	PS	-	-	-	-	-
SBLD1	U	U	U	PS	PS	-	-	PS	PS	-	-	-	-	-	-
SDEC0	-	-	-	PS	-	PT	PT	-	-	PT	T	T	T	-	-
SDEC01	-	-	-	PS	PS	PT	PT	PT	PT	PT	T	T	T	-	-
SDEC1	-	-	-	PS	PS	-	-	PT	PT	-	-	T	-	-	-
SSLV	-	-	-	PS	-	-	-	-	-	-	-	-	-	U	Ø

Table 2

<u>Usage of Arguments in the Symmetric Case</u>

(for legend see Table 3)

Table 3

Usage of Arguments in the Nonsymmetric Case

Name	I	J	V	MD	FD	RY	CY	A	AN	NDR	NDC	IPR	IPC	IE	IH	NG1	NG2	X	Y
Type	I	I	R	I	R	I	I	R	R	I	I	I	I	I	I	I	I	R	R
Length / Program	1	1	1	8	7	M_1	M_1	M_1	N	N	N	N	N	N	N	N	N	N	N
NINT01	-	-	-	US	US	S	S	-	S	S	S	-	-	-	-	-	-	-	-
NINT1	-	-	-	UPS	US	-	-	-	S	-	-	-	-	-	-	-	-	-	-
NBLD01	U	U	U	PS	PS	PS	PS	PS	PS	PS	PS	-	-	-	-	-	-	-	-
NBLD1	U	U	U	PS	PS	-	-	PS	PS	-	-	-	-	-	-	-	-	-	-
NDEC01 (PVT01)	-	-	-	PS	PS	PT	PT	PT	PT	PT	PT	T	T	T	T	T	T	-	-
NDEC1	-	-	-	PS	PS	-	-	PT	PT	-	-	-	-	T	T	-	-	-	-
NSLV	-	-	-	PS	-	-	-	-	T	-	-	-	-	-	-	-	-	U	∅

Legend: Argument type: I - integer
R - real

Entries: U - user-supplied data
S - upon exit, the argument contains data to be saved for other reasons
P - contains data generated by previously called program
T - temporary storage
∅ - output result

5. Some Experimental Results

Two groups of computational experiments were conducted on the Univac 1108 of the University of Maryland, Computer Science Center. They correspond to the computational experiments reported in [1]. Since the basic decomposition procedure is the same as in [1], the overall elapsed time for execution of the decomposition programs (T_{LU}) and the number of elements after decomposition (M_2) are essentially the same as reported there.

The new results presented below concern the maximal in-core storage requirement (M_1), the elapsed time for execution of the decomposition routines (T_1) using a previously generated decomposition record, and the elapsed time for execution of the backsubstitution routines (T_{SO}) when the decomposed matrices are residing on auxiliary storage. These times are given below relative to the elapsed time (T_{LU}) for the execution of the decomposition programs. It should be pointed out that for N larger than 100 there is less than a three percent difference between the elapsed time for the generation of a decomposition record (SDEC0) and that for a regular decomposition with or without retaining the record (SDEC01, NDEC01).

The first group of experiments involved nonsymmetric matrix decompositions. As in [1], a special program was used to generate permuted diagonally dominant random sparse matrices $B = (b_{ij})$. Given the dimension N of B and a number M_0 of nonzero elements, the program randomly generates $M_0 - N$ distinct index pairs (i,j), $i \neq j$, $1 \leq i,j \leq N$, and the corresponding matrix elements b_{ij}. Then each diagonal element is obtained by adding a random positive number to the sum of the moduli of the off-diagonal elements in its row. Finally, the rows and columns are independently and randomly

permuted. Table 4 contains the results obtained when the decomposition programs are applied to these random matrices. The pivot selection parameter $\mu = 0.125$ was used.

Table 4

Results for Nonsymmetric Random Matrices

N	M_0	M_1	M_2	T_1/T_{LU}	T_{SO}/T_{LU}
50	200	200	375	.255	.068
50	300	300	632	.218	.041
100	400	400	811	.198	.044
100	600	1,114	2,026	.140	.016
200	800	913	2,312	.138	.020
200	1,200	3,991	6,439	.074	.005
300	1,200	1,924	4,224	.121	.012

The second group of experiments involved the decomposition of symmetric matrices obtained by the discretizing of Dirichlet's problem for Laplace's equation on the unit square with the five-point and nine-point formulas. In all cases a uniform mesh was used. The coefficients of the resulting matrices are well known and are not repeated here. Table 5 contains the results obtained with the symmetric decomposition programs.

Table 5

Symmetric Matrix Decomposition

5-point Formula	M_0	M_1	M_2	T_1/T_{LU}	T_{SO}/T_{LU}
N = 81	225	225	469	.47	.18
N = 289	833	883	2,413	.22	.06
N = 484	1,408	1,699	4,733	.17	.04
N = 625	1,825	2,328	6,566	.16	.03
9-point Formula					
N = 81	353	353	715	.35	.10
N = 289	1,345	1,673	4,296	.15	.03
N = 484	2,290	2,928	8,054	.13	.02
N = 625	2,977	4,047	11,800	.10	.01

6. Program Listings

6.1 Main Package:

```
      SUBROUTINE SINT0(MD,RY,CY,ND)
      DIMENSION MD(1),RY(1),CY(1),ND(1)
      INTEGER RY,CY
C ******************************************
C * INITIALIZE SYMMETRIC STRUCTURE ARRAYS *
C ******************************************
C
      N = MD(1)
      MD(5) = N
      DO 10 I=1,N
      RY(I) = I
      CY(I) = I
10    ND(I) = 1
      RETURN
      END
```

```
      SUBROUTINE SINT01(MD,FD,RY,CY,A,AN,ND)
      DIMENSION MD(1),FD(1),RY(1),CY(1),A(1),AN(1),ND(1)
      INTEGER RY,CY
C ***********************************************************
C * INITIALIZE SYMMETRIC STRUCTURE AND COEFFICIENT ARRAYS *
C ***********************************************************
C
      N = MD(1)
      MD(5) = N
      FD(3) = 0.
      DO 10 I=1,N
      AN(I) = 0.
      RY(I) = I
      CY(I) = I
10    ND(I) = 1
      RETURN
      END
```

```
      SUBROUTINE SINT1(MD,FD,AN)
      DIMENSION MD(1),FD(1),AN(1)
C ******************************************
C * INITIALIZE SYMMETRIC COEFFICIENT ARRAY *
C ******************************************
C
      N = MD(1)
      MD(5) = N
      FD(3) = 0.
      M0 = MD(5)
      DO 10 I=1,N
10    AN(I) = 0.
      RETURN
      END
C
```

```
      SUBROUTINE NINT01(MD,FD,RY,CY,AN,NDR,NDC)
      DIMENSION MD(1),FD(1),RY(1),CY(1),NDR(1),NDC(1),AN(1)
      INTEGER RY,CY
C ***************************************************************
C * INITIALIZE NONSYMMETRIC STRUCTURE AND COEFFICIENT ARRAYS *
C ***************************************************************
C
      N = MD(1)
      MD(5) = N
      FD(3) = 0.
      DO 10 I=1,N
      AN(I) = 0.
      RY(I) = I
      CY(I) = I
      NDR(I) = 0
   10 NDC(I) = 0
      RETURN
      END
```

```
      SUBROUTINE NINT1(MD,FD,AN)
      DIMENSION MD(1),FD(1),AN(1)
C ******************************************
C * INITIALIZE NONSYMMETRIC COEFFICIENT ARRAY *
C ******************************************
C
      N = MD(1)
      MD(5) = N
      FD(3) = 0.
      DO 10 I=1,N
   10 AN(I) = 0.
      RETURN
      END
```

```
      SUBROUTINE SBLD0(I,J,MD,RY,CY,ND)
      DIMENSION MD(1),RY(1),CY(1),ND(1)
      INTEGER RY,CY
C **********************************
C * BUILD SYMMETRIC STRUCTURE ARRAY *
C **********************************
C
      IF (I.EQ.J) RETURN
      MD(5) = MD(5)+1
      MU = MD(5)
      ND(I) = ND(I)+1
      ND(J) = ND(J)+1
      IF (I.GT.J) GO TO 10
      RY(MU) = RY(I)
      CY(MU) = CY(J)
      RY(I) = MU
      CY(J) = MU
      RETURN
   10 RY(MU) = RY(J)
      CY(MU) = CY(I)
      RY(J) = MU
      CY(I) = MU
      RETURN
      END
```

```
      SUBROUTINE SBLD01(I,J,V,MD,FD,RY,CY,A,AN,ND)
      DIMENSION MD(1),FD(1),RY(1),CY(1),A(1),AN(1),ND(1)
      INTEGER RY,CY
C ****************************************************
C * BUILD SYMMETRIC STRUCTURE AND COEFFICIENT ARRAYS *
C ****************************************************
C
      S = V**2
      FD(3) = AMAX1(FD(3),ABS(V))
      AN(I) = AN(I)+S
      IF (I.EQ.J) GO TO 20
      MD(5) = MD(5)+1
      MO = MD(5)
      A(MO) = V
      AN(J) = AN(J)+S
      ND(I) = ND(I)+1
      ND(J) = ND(J)+1
      IF (I.GT.J) GO TO 10
      RY(MO) = RY(I)
      CY(MO) = CY(J)
      RY(I) = MO
      CY(J) = MO
      RETURN
   10 RY(MO) = RY(J)
      CY(MO) = CY(I)
      RY(J) = MO
      CY(I) = MO
      RETURN
   20 A(I) = V
      RETURN
      END
```

```
      SUBROUTINE SBLD1(I,J,V,MD,FD,A,AN)
      DIMENSION MD(1),FD(1),A(1),AN(1)
C *************************************
C * BUILD SYMMETRIC COEFFICIENT ARRAY *
C *************************************
C
      S = V**2
      FD(3) = AMAX1(FD(3),ABS(V))
      AN(I) = AN(I)+S
      IF (I.EQ.J) GO TO 10
      MD(5) = MD(5)+1
      MO = MD(5)
      A(MO) = V
      AN(J) = AN(J)+S
      RETURN
   10 A(I) = V
      RETURN
      END
```

```
      SUBROUTINE NBLD01(I,J,V,MD,FD,RY,CY,A,AN,NDR,NDC)
      DIMENSION MD(1),FD(1),RY(1),CY(1),A(1),NDR(1),NDC(1),AN(1)
      INTEGER RY,CY
C *********************************************************
C * BUILD NONSYMMETRIC STRUCTURE AND COEFFICIENT ARRAYS *
C *********************************************************
C
      MD(5) = MD(5)+1
      MU = MD(5)
      RY(MU) = RY(I)
      CY(MU) = CY(J)
      RY(I) = MU
      CY(J) = MU
      NDR(I) = NDR(I)+1
      NDC(J) = NDC(J)+1
      A(MU) = V
      AN(I) = AN(I)+V**2
      FD(3) = AMAX1(FD(3),ABS(V))
      RETURN
      END
```

```
      SUBROUTINE NBLD1(I,J,V,MD,FD,A,AN)
      DIMENSION MD(1),FD(1),A(1),AN(1)
C *********************************************************
C * BUILD NONSYMMETRIC COEFFICIENT ARRAY *
C *********************************************************
C
      AN(I) = AN(I)+V**2
      MD(5) = MD(5)+1
      MU = MD(5)
      A(MU) = V
      FD(3) = AMAX1(FD(3),ABS(V))
      RETURN
      END
```

```
      SUBROUTINE SDECO(MD,RY,CY,ND,IP,IE,IH)
      DIMENSION MD(1),RY(1),CY(1),ND(1),IP(1),IE(1),IH(1)
      INTEGER RY,CY
C *****************************************
C * GENERATE SYMMETRIC DECOMPOSITION RECORD *
C *****************************************
C
      MM = 0
      N = MD(1)
      MD(4) = 0
      MD(6) = MD(5)
      MD(7) = MD(5)
      MD(8) = 0
C INITIALIZE AUXILIARY FILE FOR WRITING
      CALL DWI
      DO 10 I=1,N
10    IP(I) = I
```

```
C
C  LOOP ON PIVOTING
C
         DO 250 I=1,N
         K = 0
         IF (I.EQ.N) GO TO 30
C
C  SELECT PIVOT BY MINIMAL DEGREE
C
         NDX = N+1
         DO 20 J=I,N
         IX = IP(J)
         IF (ND(IX).GE.NDX) GO TO 20
         NDX = ND(IX)
         IY = J
   20    CONTINUE
         IF (I.EQ.IY) GO TO 30
         J = IP(IY)
         IP(IY) = IP(I)
         IP(I) = J
C
C  COLLECT THE ROW AND COLUMN OF THE PIVOT
C     ALSO DELETE THEM FROM THE STORAGE
C
   30    IX = IP(I)
         IF (I.EQ.N) GO TO 110
         IY1 = 0
         IY = IX
   40    IY = RY(IY)
         IF (IY.EQ.IX) GO TO 70
         IZ = IY
   50    IZ = CY(IZ)
         IF (IZ.GT.N) GO TO 60
         K = K+1
         IE(K) = IY
         IH(K) = IZ
         ND(IZ) = ND(IZ)-1
   60    IF (CY(IZ).NE.IY) GO TO 50
         CY(IZ) = CY(IY)
         CY(IY) = MM
         MM = IY
         GO TO 40
   70    IY = CY(IY)
         IF (IY.EQ.IX) GO TO 100
         IZ = IY
         IY1 = IY
   80    IZ = RY(IZ)
         IF (IZ.GT.N) GO TO 90
         K = K+1
         IE(K) = IY
         IH(K) = IZ
         ND(IZ) = ND(IZ)-1
   90    IF (RY(IZ).NE.IY) GO TO 80
         RY(IZ) = RY(IY)
         GO TO 70
  100    IF (IY1.EQ.0) GO TO 110
         CY(IY1) = MM
         MM = CY(IX)
C
C   MODIFICATION OF THE ROW ELEMENTS
C
C        WRITE OUT PIVOT
  110    CALL DW(IX)
         CALL DW(K)
         MD(8) = MD(8)+1+K**2
         IF (K.EQ.0) GO TO 250
         DO 115 J=1,K
         CALL DW(IE(J))
  115    CALL DW(IH(J))
         IF (K.EQ.1) GO TO 250
         K1 = K-1
```

```
C
C   LOOP FOR THE CROSS-POINT ELEMENTS
C
        DO 240 J=1,K1
        J1 = J+1
        IZ = IH(J)
        DO 230 JJ=J1,K
        JZ = IH(JJ)
        I1 = MIN0(IZ,JZ)
        I2 = MAX0(IZ,JZ)
        L1 = RY(I1)
        L2 = CY(I2)
 120    IF ((L1.EQ.I1).OR.(L2.EQ.I2)) GO TO 140
        IF (L1.EQ.L2) GO TO 220
        IF (L1.GT.L2) GO TO 130
        L2 = CY(L2)
        GO TO 120
 130    L1 = RY(L1)
        GO TO 120
C   INSERTION OF A NEW NON-ZERO ELEMENT
 140    ND(I1) = ND(I1)+1
        ND(I2) = ND(I2)+1
        MD(7) = MD(7)+1
        IF (MM.NE.0) GO TO 170
C   USE NEW STORAGE
        MD(6) = MD(6)+1
        IF (MD(6).LE.MD(2)) GO TO 160
        MD(4) = 0
        RETURN
 160    L1 = MD(6)
        RY(L1) = RY(I1)
        CY(L1) = CY(I2)
        RY(I1) = L1
        CY(I2) = L1
        GO TO 220
C   USE AVAILABLE STORAGE
 170    L1 = MM
        MM = CY(MM)
        L3 = I1
        L2 = I2
 180    IF (RY(L3).LT.L1) GO TO 190
        L3 = RY(L3)
        GO TO 180
 190    RY(L1) = RY(L3)
        RY(L3) = L1
 200    IF (CY(L2).LT.L1) GO TO 210
        L2 = CY(L2)
        GO TO 200
 210    CY(L1) = CY(L2)
        CY(L2) = L1
C   WRITE OUT CROSS-POINT ELEMENT
 220    CALL DW(L1)
 230    CONTINUE
 240    CONTINUE
C   END OF MODIFICATION LOOP
 250    CONTINUE
C
C   END OF PIVOTING LOOP
C
        CALL DWE
        RETURN
C
        END
```

```
      SUBROUTINE SDEC01(MD,FD,RY,CY,A,AN,ND,IP,IE,IH)
      DIMENSION MD(1),FD(1),RY(1),CY(1),ND(1),IP(1),IE(1),IH(1)
      DIMENSION A(1),AN(1)
      INTEGER RY,CY
C ********************************************
C * DECOMPOSE SYMMETRIC MATRIX AND OPTIONALLY *
C *      GENERATE DECOMPOSITION RECORD        *
C ********************************************
C
      N = MD(1)
      MD(4) = 0
      FD(5) = 0.
      FD(6) = 1.
      FD(7) = 0.
      FD(4) = 0.
      MM = 0
      MD(6) = MD(5)
      MD(7) = MD(5)
      MD(8) = 0
C INITIALIZE AUXILIARY FILE FOR WRITING
      IF (MD(3).NE.0) CALL DWI
      DO 10 I=1,N
      FD(7) = FD(7)+ALOG(AN(I))
10    IP(I) = I
      FD(7) = 0.5*FD(7)
      CALL SVWI
C
C LOOP ON PIVOTING
C
      DO 250 I=1,N
      K = 0
      IF (I.EQ.N) GO TO 30
C
C SELECT PIVOT BY MINIMAL DEGREE
C
      NDX = N+1
      AX = 0.
      DO 15 J=I,N
      IX = IP(J)
15    AX = AMAX1(AX,ABS(A(IX)))
      AX = AX*FD(2)
      DO 20 J=I,N
      IX = IP(J)
      IF (ABS(A(IX)).LT.AX) GO TO 20
      IF (ND(IX).GE.NDX) GO TO 20
      NDX = ND(IX)
      IY = J
20    CONTINUE
      IF (I.EQ.IY) GO TO 30
      J = IP(IY)
      IP(IY) = IP(I)
      IP(I) = J
C
C COLLECT THE ROW AND COLUMN OF THE PIVOT
C    ALSO DELETE THEM FROM THE STORAGE
C
30    IX = IP(I)
      S = A(IX)
      IF (ABS(S).LT.FD(1)) GO TO 300
      IF (I.EQ.N) GO TO 110
      IY1 = 0
      IY = IX
```

```
   40    IY = RY(IY)
         IF (IY.EQ.IX) GO TO 70
         IZ = IY
   50    IZ = CY(IZ)
         IF (IZ.GT.N) GO TO 60
         K = K+1
         IE(K) = IY
         IH(K) = IZ
         AN(K) = A(IY)
         A(IY) =0.
         ND(IZ) = ND(IZ)-1
   60    IF (CY(IZ).NE.IY) GO TO 50
         CY(IZ) = CY(IY)
         CY(IY) = MM
         MM = IY
         GO TO 40
   70    IY = CY(IY)
         IF (IY.EQ.IX) GO TO 100
         IZ = IY
         IY1 = IY
   80    IZ = RY(IZ)
         IF (IZ.GT.N) GO TO 90
         K = K+1
         IE(K) = IY
         IH(K) = IZ
         AN(K) = A(IY)
         A(IY) = 0.
         ND(IZ) = ND(IZ)-1
   90    IF (RY(IZ).NE.IY) GO TO 80
         RY(IZ) = RY(IY)
         GO TO 70
  100    IF (IY1.EQ.0) GO TO 110
         CY(IY1) = MM
         MM = CY(IX)
C
C   MODIFICATION OF THE ROW ELEMENTS
C
C        WRITE OUT PIVOT
  110    IF (MD(3).NE.0) CALL DW(IX)
         IF (MD(3).NE.0) CALL DW(K)
         FD(4) = AMAX1(FD(4),ABS(S))
         FD(5) = FD(5)+ALOG(ABS(S))
         IF (S.LT.0.) FD(6) = -FD(6)
         CALL SVW(-IX,S)
         MD(8) = MD(8)+1+K**2
         IF (K.EQ.0) GO TO 250
         DO 115 J=1,K
         AN(J) = AN(J)/S
         CALL SVW(IH(J),AN(J))
         FD(4) = AMAX1(FD(4),ABS(AN(J)))
         IF (MD(3).NE.0) CALL DW(IE(J))
  115    IF (MD(3).NE.0) CALL DW(IH(J))
         K1 = K-1
C
C   LOOP FOR THE CROSS-POINT ELEMENTS
C
         DO 240 J=1,K
         J1 = J+1
         IZ = IH(J)
         Z = AN(J)
         A(IZ) = A(IZ)-S*Z**2
         FD(4) = AMAX1(FD(4),ABS(A(IZ)))
         IF (J.EQ.K) GO TO 240
```

```
      DO 230 JJ=J1,K
      JZ = IH(JJ)
      I1 = MINO(IZ,JZ)
      I2 = MAXO(IZ,JZ)
      L1 = RY(I1)
      L2 = CY(I2)
120   IF ((L1.EQ.I1).OR.(L2.EQ.I2)) GO TO 140
      IF (L1.EQ.L2) GO TO 220
      IF (L1.GT.L2) GO TO 130
      L2 = CY(L2)
      GO TO 120
130   L1 = RY(L1)
      GO TO 120
C   INSERTION OF A NEW NON-ZERO ELEMENT
140   ND(I1) = ND(I1)+1
      ND(I2) = ND(I2)+1
      MD(7) = MD(7)+1
      IF (MM.NE.0) GO TO 170
C   USE NEW STORAGE
      MD(6) = MD(6)+1
      IF (MD(6).GT.MD(2)) GO TO 310
160   L1 = MD(6)
      A(L1) = 0.
      RY(L1) = RY(I1)
      CY(L1) = CY(I2)
      RY(I1) = L1
      CY(I2) = L1
      GO TO 220
C   USE AVAILABLE STORAGE
170   L1 = MM
      MM = CY(MM)
      L3 = I1
      L2 = I2
180   IF (RY(L3).LT.L1) GO TO 190
      L3 = RY(L3)
      GO TO 180
190   RY(L1) = RY(L3)
      RY(L3) = L1
200   IF (CY(L2).LT.L1) GO TO 210
      L2 = CY(L2)
      GO TO 200
210   CY(L1) = CY(L2)
      CY(L2) = L1
C   WRITE OUT CROSS-POINT ELEMENT
220   IF (MD(3).NE.0) CALL DW(L1)
      A(L1) = A(L1)-AN(J)*AN(JJ)*S
230   FD(4) = AMAX1(FD(4),ABS(A(L1)))
240   CONTINUE
C   END OF MODIFICATION LOOP
250   CALL SVW(-IX,S)
C
C   END OF PIVOTING LOOP
C
      CALL SVWE
      IF (MD(3).NE.0) CALL DWE
      RETURN
C
C   SINGULAR MATRIX
C
300   MD(4) = 1
      RETURN
310   MD(4) = 3
      RETURN
C
      END
```

```
      SUBROUTINE SDEC1(MD,FD,A,AN,IE)
      DIMENSION MD(1),FD(1),A(1),AN(1),IE(1)
C ***********************************************************
C * DECOMPOSE SYMMETRIC MATRIX USING GENERATED RECORD *
C ***********************************************************
C
      N = MD(1)
      MD(4) = 0
      FD(5) = 0.
      FD(6) = 1.
      FD(7) = 0.
      FD(4) = 0.
      DO 10 I=1,N
10    FD(7) = FD(7)+ALOG(AN(I))
      FD(7) = 0.5*FD(7)
C CLEAR STORAGE TO BE FILLED
      IF (MD(6).LE.MD(5)) GO TO 30
      MM = MD(5)+1
      M1 = MD(6)
      DO 20 I=MM,M1
20    A(I) = 0.
C INITIALIZE FOR READ-IN AND WRITE-OUT
30    CALL DRI
      CALL SVWI
C
C LOOP ON THE PIVOTS
C
      DO 90 I=1,N
      CALL DR(IX)
      CALL DR(K)
      S = A(IX)
      IF (ABS(S).LT.FD(1)) GO TO 100
      FD(4) = AMAX1(FD(4),ABS(S))
      FD(5) = FD(5)+ALOG(ABS(S))
      IF (S.LT.0.) FD(6) = -FD(6)
      IF (K.EQ.0) GO TO 70
C
C COLLECT THE ROW OF THE PIVOT
C
      DO 40 J=1,K
      CALL DR(L1)
      CALL DR(IE(J))
      AN(J) = A(L1)
      A(L1) = 0.
40    FD(4) = AMAX1(FD(4),ABS(AN(J)))
C
C   MODIFICATION OF THE ROW ELEMENTS
C
      DO 60 J=1,K
      IZ = IE(J)
      Z = AN(J)
      AN(J) = AN(J)/S
      A(IZ) = A(IZ)-Z*AN(J)
      FD(4) = AMAX1(FD(4),ABS(AN(J)))
      FD(4) = AMAX1(FD(4),ABS(A(IZ)))
      IF (J.EQ.K) GO TO 60
      J1 = J+1
C
C   MODIFICATION OF THE CROSS-POINT ELEMENTS
C
      DO 50 JJ=J1,K
      CALL DR(L1)
      A(L1) = A(L1)-AN(J)*AN(JJ)
50    FD(4) = AMAX1(FD(4),ABS(A(L1)))
60    CONTINUE
```

```
C
C WRITE OUT PIVOT AND ITS ROW
C
70      CALL SVW(-IX,S)
        IF (K.EQ.0) GO TO 90
        DO 80 J=1,K
80      CALL SVW(IE(J),AN(J))
90      CALL SVW(-IX,S)
C
C END OF PIVOTING LOOP
C
        CALL SVWE
        RETURN
C
C SINGULAR MATRIX
C
100     MD(4) = 3
        RETURN
C
        END
```

```
        SUBROUTINE NDEC01(MD,FD,RY,CY,A,AN,NDR,NDC,IPR,IPC,IE,IH,NG1,NG2)
        DIMENSION MD(1),FD(1),RY(1),CY(1),A(1),AN(1),NDR(1),NDC(1)
        DIMENSION IPR(1),IPC(1),IE(1),IH(1),NG1(1),NG2(1)
        INTEGER RY,CY
C ***********************************************
C * DECOMPOSE NONSYMMETRIC MATRIX AND OPTIONALLY *
C *      GENERATE DECOMPOSITION RECORD            *
C ***********************************************
C
        MM = 0
        N = MD(1)
        MD(4) = 0
        MD(6) = MD(5)
        MD(7) = MD(5)
        MD(8) = 0
        IF (MD(3).NE.0) CALL DWI
        FD(5) = 0.
        FD(6) = 1.
        FD(7) = 0.
        FD(4) = 0.
        DO 10 I=1,N
        FD(7) = FD(7)+ALOG(AN(I))
        IPR(I) = I
10      IPC(I) = I
        FD(7) = 0.5*FD(7)
        CALL NVWI
        CALL NVWB(-1,-1)
C
C LOOP ON PIVOTING
C
        DO 290 I=1,N
C
C  PIVOT SELECTION BY SEPARATE PIVOTING ROUTINE
C
        CALL PVT01(I,N,IX,KR,KC,FD(2),RY,CY,A,IPR,IPC,NDR,NDC,IE,IH)
        IF (KR.EQ.0) GO TO 310
C  CHECK PIVOT VALUE AND UPDATE DETERMINANT
        S = A(IX)
        IF (ABS(S).LE.FD(1)) GO TO 300
        FD(5) = FD(5)+ALOG(ABS(S))
        K = (KR+KC)/2
        K = KR+KC-2*K
        IF (K.NE.0) FD(6) = -FD(6)
        FD(4) = AMAX1(FD(4),ABS(S))
```

```
C
C   COLLECT THE ROW AND COLUMN OF THE PIVOT
C   ALSO FREE THEIR STORAGE LOCATIONS
C
C   THE ROW
          K1 = 0
          J = KR
 20       J = RY(J)
          IF (J.LE.N) GO TO 40
          IF (J.EQ.IX) GO TO 20
          K1 = K1+1
          IE(K1) = J
          AN(K1) = A(J)
          J1 = J
 30       J1 = CY(J1)
          IF (J1.LE.N) NG1(K1) = J1
          IF (J1.LE.N) NDC(J1) = NDC(J1)-1
          IF (CY(J1).NE.J) GO TO 30
          CY(J1) = CY(J)
          CY(J) = MM
          MM = J
          GO TO 20
C   THE COLUMN
 40       K2 = 0
          J = KC
          K3 = 0
 50       J = CY(J)
          IF (J.LE.N) GO TO 70
          K3 = J
          IF (J.EQ.IX) GO TO 50
          K2 = K2+1
          IH(K2) = J
          A(K2) = A(J)/S
          FD(4) = AMAX1(FD(4),ABS(A(K2)))
          J1 = J
 60       J1 = RY(J1)
          IF (J1.LE.N) NG2(K2) = J1
          IF (J1.LE.N) NDR(J1) = NDR(J1)-1
          IF (RY(J1).NE.J) GO TO 60
          RY(J1) = RY(J)
          GO TO 50
 70       CY(K3) = MM
          MM = CY(KC)
C
C   WRITE OUT THE PIVOT, ITS ROW AND COLUMN
C   AS PART OF THE DECOMPOSITION RECORD
C
          IF (MD(3).EQ.0) GO TO 105
          CALL DW(IX)
          CALL DW(KC)
          CALL DW(KR)
          CALL DW(K2)
          MD(8) = MD(8)+(K1+1)*(K2+1)
          IF (K2.EQ.0) GO TO 90
          DO 80 J=1,K2
          CALL DW(IH(J))
 80       CALL DW(NG2(J))
 90       CALL DW(K1)
          IF (K1.EQ.0) GO TO 105
          DO 100 J=1,K1
          CALL DW(IE(J))
 100      CALL DW(NG1(J))
 105      IF ((K1.EQ.0).OR.(K2.EQ.0)) GO TO 230
C
C   LOOP TO MODIFY THE INTERSECTING ELEMENTS BETWEEN THE ROW
C            AND COLUMN
C
```

```
          DO 220 J=1,K2
          IY = NG2(J)
          DO 220 K=1,K1
          K3 = RY(IY)
          JY = NG1(K)
          K4 = CY(JY)
C   SEARCH FOR ELEMENT IY,JY
110       IF (K3.EQ.K4) GO TO 210
          IF (K3.LT.K4) GO TO 120
          K3 = RY(K3)
          IF (K3.LE.N) GO TO 130
          GO TO 110
120       K4 = CY(K4)
          IF (K4.GT.N) GO TO 110
C   IT DOES NOT EXIST
130       NDR(IY) = NDR(IY)+1
          NDC(JY) = NDC(JY)+1
          MD(7) = MD(7)+1
          IF (MM.NE.0) GO TO 160
C    GET NEW LOCATION FOR THE NEW ELEMENT
          MD(6) = MD(6)+1
          IF (MD(6).LE.MD(2)) GO TO 150
          MD(4) = 1
          RETURN
150       K3 = MD(6)
          RY(K3) = RY(IY)
          CY(K3) = CY(JY)
          RY(IY) = K3
          CY(JY) = K3
          A(K3) = 0.
          GO TO 210
C    OLD LOCATION AVAILABLE FOR THE NEW ELEMENT
160       K3 = MM
          A(K3) = 0.
          MM = CY(MM)
          K4 = IY
170       IF (RY(K4).LT.K3) GO TO 180
          K4 = RY(K4)
          GO TO 170
180       RY(K3) = RY(K4)
          RY(K4) = K3
190       IF (CY(JY).LT.K3) GO TO 200
          JY = CY(JY)
          GO TO 190
200       CY(K3) = CY(JY)
          CY(JY) = K3
C    MODIFY ELEMENT
210       IF (MD(3).NE.0) CALL DW(K3)
          A(K3) = A(K3)-AN(K)*A(J)
          FD(4) = AMAX1(FD(4),ABS(A(K3)))
220       CONTINUE
C   END OF MODIFICATION LOOP
C
C WRITE OUT PIVOT ROW AND COLUMN
C
230       CALL NVWF(-KC,-KR)
          IF (K2.EQ.0) GO TO 250
          DO 240 J=1,K2
240       CALL NVWF(NG2(J),A(J))
250       IF (K1.EQ.0) GO TO 270
          DO 260 J=1,K1
260       CALL NVWB(NG1(J),AN(J))
270       CALL NVWB(-KC,S)
290       CONTINUE
C   END OF PIVOTING LOOP
C
C END FILES
```

```
C
      IF (MD(3).NE.0) CALL DWE
      CALL NVWE
      RETURN
C
C SINGULAR MATRIX
C
300   MD(4) = 3
      RETURN
310   MD(4) = 2
      RETURN
C
      END
```

```
      SUBROUTINE PVT01(I,N,IX,KR,KC,F,RY,CY,A,IPR,IPC,NDR,NDC,IE,IH)
      DIMENSION RY(1),CY(1),IPR(1),IPC(1),NDR(1),NDC(1),IE(1),IH(1)
      DIMENSION A(1)
      INTEGER RY,CY
C ***************************************************************
C * MINIMAL DEGREE PIVOTING FOR NON-SYMMETRIC MATRIX *
C ***************************************************************
C     THE ROUTINE SELECTS PIVOT BY MINIMAL DEGREE, IT IS
C USED BY THE ROUTINE DEC01.
C
C
      IF (I.NE.N) GO TO 30
      KR = IPR(N)
      KC = IPC(N)
      IX = RY(KR)
      IF (IX.NE.KR) RETURN
10    KR = 0
      RETURN
30    NI = N+1-I
C
C SORT AVAILABLE ROWS BY DEGREE
C
      DO 40 J=1,NI
40    IE(J) = 0
      DO 50 J=I,N
      K1 = IPR(J)
      IH(J) = K1
      K2 = NDR(K1)
      IF (K2.LE.0) GO TO 10
50    IE(K2) = IE(K2)+1
      DO 60 J=2,NI
60    IE(J) = IE(J)+IE(J-1)
      DO 70 J=I,N
      K1 = IH(J)
      K2 = NDR(K1)
      K3 = IE(K2)+I-1
      IE(K2) = IE(K2)-1
70    IPR(K3) = K1
C
C SORT AVAILABLE COLUMNS BY DEGREE
C
      DO 80 J=1,NI
80    IE(J) = 0
      DO 90 J=I,N
      K1 = IPC(J)
      IH(J) = K1
      K2 = NDC(K1)
      IF (K2.LE.0) GO TO 10
90    IE(K2) = IE(K2)+1
```

```
      DO 100 J=2,NI
100   IE(J) = IE(J)+IE(J-1)
      DO 110 J=1,N
      K1 = IH(J)
      K2 = NDC(K1)
      K3 = IE(K2)+I-1
      IE(K2) = IE(K2)-1
110   IPC(K3) = K1
C
C INITIALIZE FOR MINIMAL DEGREE SEARCH
C
      IX = 0
      IDX = N**2
      JR = I
      JC = I
      JRP = IPR(JR)
      JCP = IPC(JC)
C
C TEST FOR TERMINATION OF SEARCH
C
120   NDX = (NDR(JRP)-1)*(NDC(JCP)-1)
      IF (NDX.GE.IDX) GO TO 240
      IF (NDC(JCP).GT.NDR(JRP)) GO TO 180
C
C SEARCH IN THE COLUMN
C
      J = JCP
      AM = 0.
130   J = CY(J)
      IF (J.EQ.JCP) GO TO 140
      AM = AMAX1(AM,ABS(A(J)))
      GO TO 130
140   AM = F*AM
150   J = CY(J)
      IF (J.EQ.JCP) GO TO 170
      IF (ABS(A(J)).LT.AM) GO TO 150
      K1 = J
160   K1 = RY(K1)
      IF (K1.GT.N) GO TO 160
      K2 = (NDR(K1)-1)*(NDC(JCP)-1)
      IF (K2.GE.IDX) GO TO 150
      IX = J
      KR = K1
      KC = JCP
      IDX = K2
      GO TO 150
170   JC = JC+1
      IF (JC.GT.N) GO TO 240
      JCP = IPC(JC)
      GO TO 120
C
C SEARCH IN THE ROW
C
180   J = JRP
      AM = 0.
190   J = RY(J)
      IF (J.EQ.JRP) GO TO 200
      AM = AMAX1(AM,ABS(A(J)))
      GO TO 190
200   AM = F*AM
210   J = RY(J)
      IF (J.EQ.JRP) GO TO 230
      IF (ABS(A(J)).LT.AM) GO TO 210
      K1 = J
220   K1 = CY(K1)
      IF (K1.GT.N) GO TO 220
```

```
      K2 = (NDR(JRP)-1)*(NDC(K1)-1)
      IF (K2.GE.IDX) GO TO 210
      IX = J
      KR = JRP
      KC = K1
      IDX = K2
      GO TO 210
230   JR = JR+1
      IF (JR.GT.N) GO TO 240
      JRP = IPR(JR)
      GO TO 120
C
C SEARCH FINISHED, REMOVE KR,KC FROM AVAILABLE
C PIVOT ROWS AND COLUMNS
240   IF (IX.EQ.0) GO TO 10
      DO 250 J=I,N
      IF (IPR(J).NE.KR) GO TO 250
      IF (J.EQ.I) GO TO 260
      IPR(J) = IPR(I)
      IPR(I) = KR
      GO TO 260
250   CONTINUE
260   DO 270 J=I,N
      IF (IPC(J).NE.KC) GO TO 270
      IPC(J) = IPC(I)
      IPC(I) = KC
      GO TO 280
270   CONTINUE
C
280   RETURN
      END
```

```
      SUBROUTINE NDEC1(MD,FD,A,AN,IE,IH)
      DIMENSION MD(1),FD(1),A(1),AN(1),IE(1),IH(1)
C ***********************************************************
C * DECOMPOSE NONSYMMETRIC MATRIX USING GENERATED RECORD *
C ***********************************************************
C
      N = MD(1)
      MD(4) = 0
      FD(5) = 0.
      FD(6) = 1.
      FD(7) = 0.
      FD(4) = 0.
      DO 10 I=1,N
10    FD(7) = FD(7)+ALOG(AN(I))
      FD(7) = 0.5*FD(7)
C CLEAR EXTRA STORAGE
      IF (MD(6).LE.MD(5)) GO TO 30
      MM = MD(5)+1
      M1 = MD(6)
      DO 20 I=MM,M1
20    A(I) = 0.
```

```
      C INITIALIZE FILES
      30    CALL DRI
            CALL NVWI
            CALL NVWB(-1,-1)
      C
      C LOOP ON THE PIVOTS
      C
            DO 130 I=1,N
      C GET PIVOT ADDRESS AND CHECK PIVOT MAGNITUDE
            CALL DR(IX)
            S = A(IX)
            IF (ABS(S).LE.FD(1)) GO TO 150
            FD(5) = FD(5)+ALOG(ABS(S))
            IF (S.LT.0.) FD(6) = -FD(6)
            FD(4) = AMAX1(FD(4),ABS(S))
            A(IX) = 0.
            CALL DR(KC)
            CALL DR(KR)
      C GET THE COLUMN ELEMENTS
            CALL DR(K2)
            IF (K2.LE.0) GO TO 50
            DO 40 J=1,K2
            CALL DR(K3)
            CALL DR(IE(J))
            A(J) = A(K3)/S
            FD(4) = AMAX1(FD(4),ABS(A(J)))
      40    A(K3) = 0.
      C GET THE ROW ELEMENTS
      50    CALL DR(K1)
            IF (K1.LE.0) GO TO 80
            DO 60 J=1,K1
            CALL DR(K3)
            CALL DR(IH(J))
            AN(J) = A(K3)
            FD(4) = AMAX1(FD(4),ABS(AN(J)))
      60    A(K3) = 0.
      C MODIFY CROSS-POINT ELEMENTS
            IF (K2.LE.0) GO TO 80
            DO 70 J=1,K2
            DO 70 JJ=1,K1
            CALL DR(K3)
            A(K3) = A(K3)-A(J)*AN(JJ)
      70    FD(4) = AMAX1(FD(4),ABS(A(K3)))
      C WRITE OUT PIVOT ROW AND COLUMN
      80    CALL NVWF(-KC,-KR)
            IF (K2.EQ.0) GO TO 100
            DO 90 J=1,K2
      90    CALL NVWF(IE(J),A(J))
      100   IF (K1.EQ.0) GO TO 120
            DO 110 J=1,K1
      110   CALL NVWB(IH(J),AN(J))
      120   CALL NVWB(-KC,S)
      130   CONTINUE
      C
            CALL NVWE
            RETURN
      C
      150   MD(4) = 3
            RETURN
      C
            END
```

```
      SUBROUTINE SSLV(MD,X,Y)
      DIMENSION MD(1),X(1),Y(1)
C ***************************************************
C * BACKSUBSTITUTION FOR SYMMETRIC DECOMPOSED MATRIX *
C ***************************************************
C
      N = MD(1)
      DO 10 I=1,N
10    Y(I) = X(I)
      CALL SVRI
      I = 0
C FORWARD BACKSUBSTITUTION
20    CALL SVRF(L2,Z)
      I = I+1
      L2 = -L2
      S = Y(L2)
30    CALL SVRF(L2,Z)
      IF (L2.LT.0) GO TO 40
      Y(L2) = Y(L2)-Z*S
      GO TO 30
40    IF (I.LT.N) GO TO 20
C BACKWARD BACKSUBSTITUTION
50    CALL SVRB(L2,Z)
      I = I-1
      J = -L2
      Y(J) = Y(J)/Z
60    CALL SVRB(L2,Z)
      IF (L2.LT.0) GO TO 70
      Y(J) = Y(J)-Z*Y(L2)
      GO TO 60
70    IF (I.GT.0) GO TO 50
      RETURN
C
      END
```

```
      SUBROUTINE NSLV(MD,X,Y,AN)
      DIMENSION MD(1),X(1),Y(1),AN(1)
C ******************************************************
C * BACKSUBSTITUTION FOR NONSYMMETRIC DECOMPOSED MATRIX *
C ******************************************************
C
      EQUIVALENCE (KS,S)
      N = MD(1)
C
C   SAVE RIGHT SIDE
C
      DO 10 I=1,N
10    AN(I) = X(I)
C
C   INITIALIZE FILES
C
      CALL NVRI
      I = 0
      J = 0
C
C   SOLVE LOWER TRIANGULAR SYSTEM
```

```
C
20     CALL NVRF(K4,S)
       IF (K4.GT.0) GO TO 30
       I = I+1
       K4 = -K4
       K3 = -KS
       Y(K4) = AN(K3)
       IF (I.GE.N) GO TO 40
       D1 = AN(K3)
       GO TO 20
30     AN(K4) = AN(K4)-D1*S
       GO TO 20
C
C   SOLVE UPPER TRIANGULAR SYSTEM
C
40     CALL NVRB(K4,S)
       IF (K4.GT.0) GO TO 50
       J = J+1
       IF (J.NE.1) Y(IX) = Y(IX)/D1
       IF (J.GT.N) GO TO 60
       IX = -K4
       D1 = S
       GO TO 40
50     Y(IX) = Y(IX)-S*Y(K4)
       GO TO 40
C
60     RETURN
C
       END
```

6.2 I/O Programs:

```
C ***********************************************
C * I/O ROUTINE FOR DECOMPOSITION PROCESSES *
C ***********************************************
C
      SUBROUTINE DWI
C
C THE DECOMPOSITION PROCESS GENERATES AN ARRAY OF
C POSITIVE INTEGERS, THIS PROGRAM PROVIDES A BUFFERED
C INPUT/OUTPUT USING FILE 10. THE ENTRIES ARE AS FOLLOWS:
C
C    DWI - INITIALIZE FOR WRITE
C    DW(K) - WRITE K AS NEXT ENTRY
C    DWE - TERMINATE WRITING
C    DRI - INITIALIZE FOR READ
C    DR(K) - READ NEXT ENTRY K
C
C IX IS THE BUFFER SIZE
      PARAMETER IX = 250
      DIMENSION IB(IX)
C
C
C ***************************
C * INITIALIZE FOR WRITING *
C ***************************
C
      J = 1
      REWIND 10
      RETURN
C
      ENTRY DW(K)
C ***********************
C * WRITE NEXT ENTRY K *
C ***********************
C
      IB(J) = K
      J = J+1
      IF (J.LE.IX) RETURN
      WRITE (10) IB
      J = 1
      RETURN
C
      ENTRY DWE
C *********************
C * TERMINATE WRITING *
C *********************
C
      IF (J.NE.1) WRITE (10) IB
      RETURN
C
      ENTRY DRI
C **********************
C * INITIALIZE READ-IN *
C **********************
C
      REWIND 10
      J = IX
      RETURN
C
      ENTRY DR(K)
C **********************
C * READ NEXT ENTRY K *
C **********************
C
      J = J+1
      IF (J.LE.IX) GO TO 10
      READ (10) IB
      J = 1
10    K = IB(J)
      RETURN
C
      END
```

```
C ***********************************************
C * I/O ROUTINE FOR DECOMPOSED SYMMETRIC MATRIX *
C ***********************************************
C
      SUBROUTINE SVWI
C THE DECOMPOSED MATRIX IS PLACED IN FILE 11 AS A RANDOM
C ACCESS FILE. IT CONSISTS OF A DOUBLE ARRAY WHICH IS
C BUFFERED. ALTHOUGH THE FIRST PART OF THE ARRAY
C IS AN INTEGER (SIGNED) ARRAY, THIS ROUTINE DOES NOT
C PACK IT. THE ROUTINE HAS THE FOLLOWING ENTRIES:
C
C    SVWI - INITIALIZE FOR WRITE
C    SVW(A1,A2) - WRITE A1,A2 AS NEXT ENTRY
C    SVWE - TERMINATE WRITE
C    SVRI - INITIALIZE FOR READ
C    SVRF(A1,A2) - READ NEXT ENTRY A1,A2
C    SVRB(A1,A2) - READ PREVIOUS ENTRY A1,A2
C
C THE ROUTINE ASSUMES THAT THE WRITTEN ARRAY IS READ ONCE
C FORWARD THEN READ BACKWARD.
C
      PARAMETER NX = 100
      PARAMETER MX = 280
      PARAMETER MXX = 2*MX
C NX IS THE MAXIMUM NUMBER OF RECORDS
C MXX IS THE LENGTH OF THE RECORDS
      DIMENSION B(2,MX)
C
C **********************
C * INITIALIZE WRITING *
C **********************
C
      N = 0
      J = 1
      DEFINE FILE 11(NX,MXX,U,IX)
      IX = IX
      RETURN
C
      ENTRY SVW(A1,A2)
C **************************
C * WRITE NEXT ENTRY A1,A2 *
C **************************
C
      B(1,J) = A1
      B(2,J) = A2
      J = J+1
      IF (J.LE.MX) RETURN
      N = N+1
      WRITE (11'N) B
      J = 1
      RETURN
C
      ENTRY SVWE
C *********************
C * TERMINATE WRITING *
C *********************
C
      IF (J.EQ.1) RETURN
      N = N+1
      WRITE (11'N) B
      RETURN
C
      ENTRY SVRI
C **********************
C * INITIALIZE READ-IN *
C **********************
C
      M = 0
      J = MX
      RETURN
```

```
C
      ENTRY SVRF(A1,A2)
C ************************
C * READ NEXT ENTRY A1,A2 *
C ************************
C
      J = J+1
      IF (J.LE.MX) GO TO 10
      M = M+1
      READ (11'M) B
      J = 1
10    A1 = B(1,J)
      A2 = B(2,J)
      RETURN
C
      ENTRY SVRB(A1,A2)
C ****************************
C * READ PREVIOUS ENTRY A1,A2 *
C ****************************
C
      IF (J.GT.0) GO TO 20
      M = M-1
      READ (11'M) B
      J = MX
20    A1 = B(1,J)
      A2 = B(2,J)
      J = J-1
      RETURN
C
      END
```

```
C ****************************************************
C * I/O ROUTINE FOR NONSYMMETRIC DECOMPOSED MATRIX *
C ****************************************************
      SUBROUTINE NVWI
C THE LOWER AND UPPER TRIANGULAR MATRICES OF THE
C DECOMPOSED NONSYMMETRIC MATRIX ARE CONTAINED IN
C FILE 12 AND FILE 13, RESPECTIVELY, AS RANDOM
C ACCESS FILES. THEY ARE IN THE FORM OF BUFFERED
C DOUBLE ARRAYS. THE ENTRIES ARE AS FOLLOWS,
C
C    NVWI - INITIALIZE FOR WRITE
C    NVWF(A1,A2) - WRITE A1,A2 AS NEXT ENTRY ON FILE 12
C    NVWB(A1,A2) - WRITE A1,A2 AS NEXT ENTRY ON FILE 13
C    NVWE - TERMINATE WRITING
C    NVRI - INITIALIZE FOR READ
C    NVRF(A1,A2) - READ NEXT ENTRY FROM FILE 12
C    NVRB(A1,A2) - READ PREVIOUS ENTRY FROM FILE 13
C
C FILE 12 IS READ FORWARD, FILE 13 BACKWARD.
C
      PARAMETER NX = 100
      PARAMETER MX = 280
      PARAMETER MXX = 2*MX
C NX IS THE MAXIMUM NUMBER OF RECORDS,
C MXX IS THE RECORD SIZE
      DIMENSION B12(2,MX),B13(2,MX)
C
C **********************
C * INITIALIZE WRITING *
C **********************
C
      N12 = 0
      N13 = 0
      J12 = 1
      J13 = 1
      DEFINE FILE 12(NX,MXX,U,IX12)
      IX12 = IX12
      DEFINE FILE 13(NX,MXX,U,IX13)
      IX13 = IX13
      RETURN
C
      ENTRY NVWF(A1,A2)
C **************************
C * WRITE A1,A2 ON FILE 12 *
C **************************
C
      B12(1,J12) = A1
      B12(2,J12) = A2
      J12 = J12+1
      IF (J12.LE.MX) RETURN
      N12 = N12+1
      J12 = 1
      WRITE (12'N12) B12
      RETURN
C
      ENTRY NVWB(A1,A2)
C **************************
C * WRITE A1,A2 ON FILE 13 *
C **************************
C
      B13(1,J13) = A1
      B13(2,J13) = A2
      J13 = J13+1
      IF (J13.LE.MX) RETURN
      N13 = N13+1
      J13 = 1
      WRITE (13'N13) B13
      RETURN
C
```

```
            ENTRY NVWE
C *********************
C * TERMINATE WRITING *
C *********************
C
            IF (J12.EQ.1) GO TO 10
            N12 = N12+1
            WRITE (12'N12) B12
10          JJ = MX
            IF (J13.EQ.1) RETURN
            N13 = N13+1
            WRITE (13'N13) B13
            JJ = J13-1
            RETURN
C
            ENTRY NVRI
C **********************
C * INITIALIZE READ-IN *
C **********************
C
            N2 = 0
            N3 = N13+1
            J2 = MX
            J3 = 0
            RETURN
C
            ENTRY NVRF(A1,A2)
C ****************************
C * READ A1,A2 FROM FILE 12 *
C ****************************
C
            J2 = J2+1
            IF (J2.LE.MX) GO TO 20
            N2 = N2+1
            READ (12'N2) B12
            J2 = 1
20          A1 = B12(1,J2)
            A2 = B12(2,J2)
            RETURN
C
            ENTRY NVRB(A1,A2)
C *******************************************
C * READ PREVIOUS ENTRY A1,A2 FROM FILE 13 *
C *******************************************
C
            IF (J3.GT.0) GO TO 30
            N3 = N3-1
            READ (13'N3) B13
            J3 = MX
            IF (N3.EQ.N13) J3 = JJ
30          A1 = B13(1,J3)
            A2 = B13(2,J3)
            J3 = J3-1
            RETURN
C
            END
```

References

[1] Rheinboldt, W.C., and Mesztenyi, C.K., "Programs for the solution of large sparse matrix problems based on the arc-graph structure", University of Maryland, Computer Science Technical Report TR-262, 1973.

[2] Rheinboldt, W.C., and Mesztenyi, C.K., "Arc graphs and their possible application to sparse matrix problems", BIT 14, 1974, 227-239.

[3] Gustavson, F.G., Liniger, W.M., and Willoughby, R.A., "Symbolic generation of an optimal Crout algorithm for sparse systems of linear equations", in Sparse Matrix Proceedings, (R.A. Willoughby, Ed.), IBM Research, Yorktown Heights, N.Y., 1968, 1-9.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER Technical Report TR-394	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Further Programs for the Solution of Large Sparse Systems of Linear Equations		5. TYPE OF REPORT & PERIOD COVERED Technical Report
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) C. K. Mesztenyi W. C. Rheinboldt		8. CONTRACT OR GRANT NUMBER(s) N00014-67-A-0239-0021 GJ-35568X
9. PERFORMING ORGANIZATION NAME AND ADDRESS Computer Science Center University of Maryland College Park, Maryland 20742		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Mathematics Branch Office of Naval Research Arlington, VA 22217		12. REPORT DATE August 1975
		13. NUMBER OF PAGES 50
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for Public Release; Distribution Unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

sparse linear systems
triangular systems
FORTRAN programs
fill-in
decomposition records

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A package of FORTRAN subroutines is presented for the solution of non-symmetric or symmetric sparse linear systems by triangular decomposition. Two principal aims are (1) to handle matrices which originally fit into primary core storage but do so no longer after decomposition, and (2) to solve a sequence of linear systems all of which have the same sparsity structure by generating--in secondary storage--a record of the decomposition process in the form of an integer array. Some experimental results using the package are included.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)